Intro to Computer Audio

→ "Computer audio" covers a large range of topics:
  → Acoustics
  → Signal Processing
  → Audio synthesis
  → Electronic Music
  → Analog / Digital Conversion
  → Misc... other...

→ Will try to cover essentials:
  ↳ ~~This~~ is a 400 level class
  → Usually we approach this with breadth
  → Will strive instead for depth.

Expected:
  → A basic background in mathematics:
    → Algebra
    → ~~Trig.~~
    → A little bit of calculus
  → Background in a C-like language
    → Will be using mostly Python for class examples
    → You may use whatever language you like

→ Review course handbook

Acoustics - the science of sound

Acoustics - the science of sound

→ Specifically, acoustics is the study of mechanical waves.

↳ Wait, what's a wave?

Definition: the motion of a disturbance (Serway / Vuille; College Physics)

→ That's pretty cryptic.

→ Mechanical waves

↳ A disturbance in a medium

→ Causes energy to propagate

→ The medium does not move in the direction of the wave.

→ The canonical example: a pebble in a pond

↳ Dropping a pebble in a pond causes a disturbance in the surface of the water.

→ The ripple moves outwards, but the water molecules move up and down.

→ A mechanical wave does not exist without a medium

→ Side note: electromagnetic waves DO

↳ Light travels through space, sound does not.

→ Sorry, star wars!

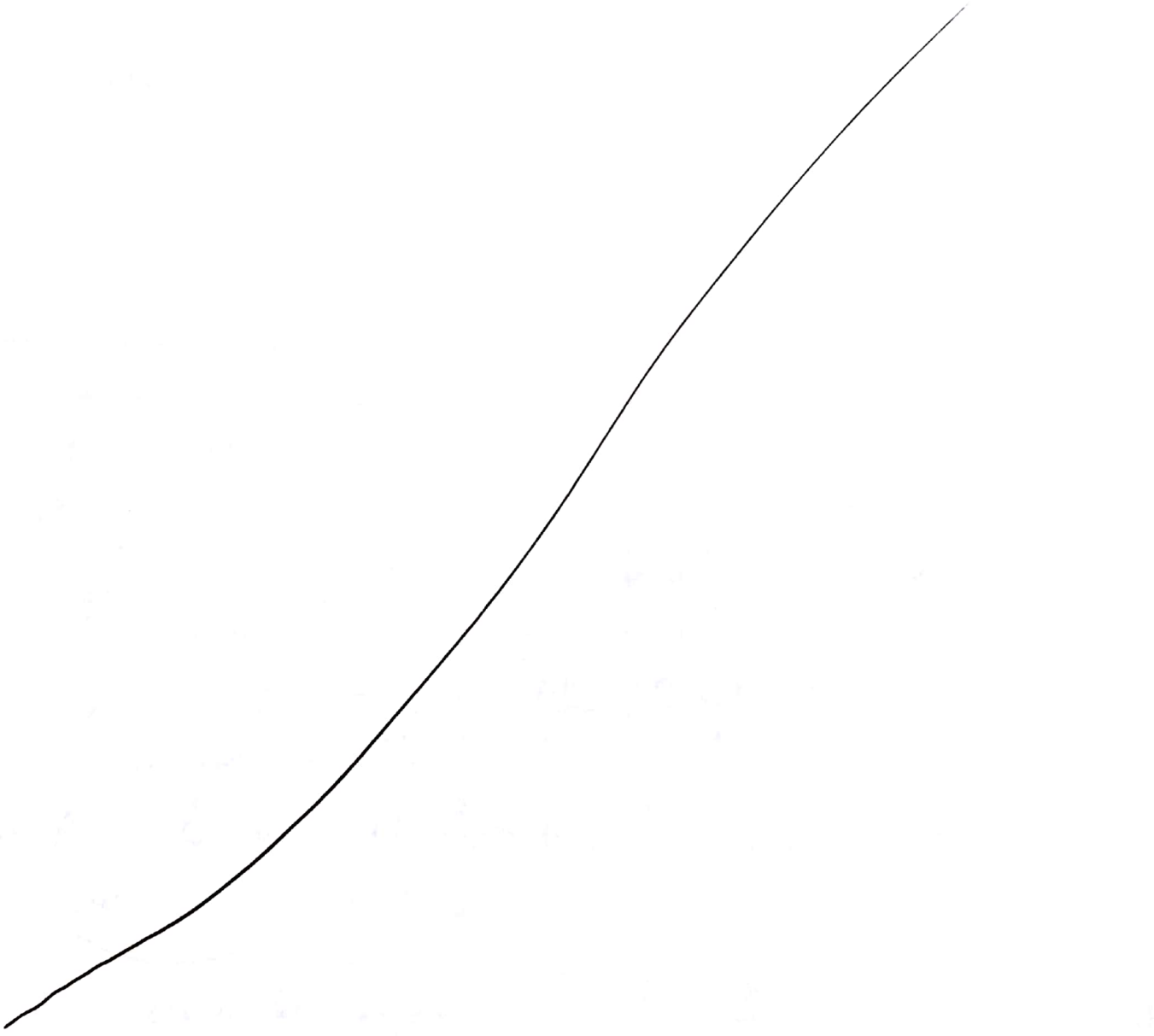
Requirements for Mechanical Waves:

1) A source of disturbance

2) A disturbable medium

3) Physical Connection between elements in a medium.

→ What is a Sound Wave?
  ↳ A Mechanical Wave that is "audible"
  → An object that vibrates generates variation in air pressure
    ↳ Air Molecules bump into each other
    → Eventually, Molecules bump into your eardrum
    → Your brain interprets these vibrations as Beethoven or Biebs
      ↳ We do not understand this!

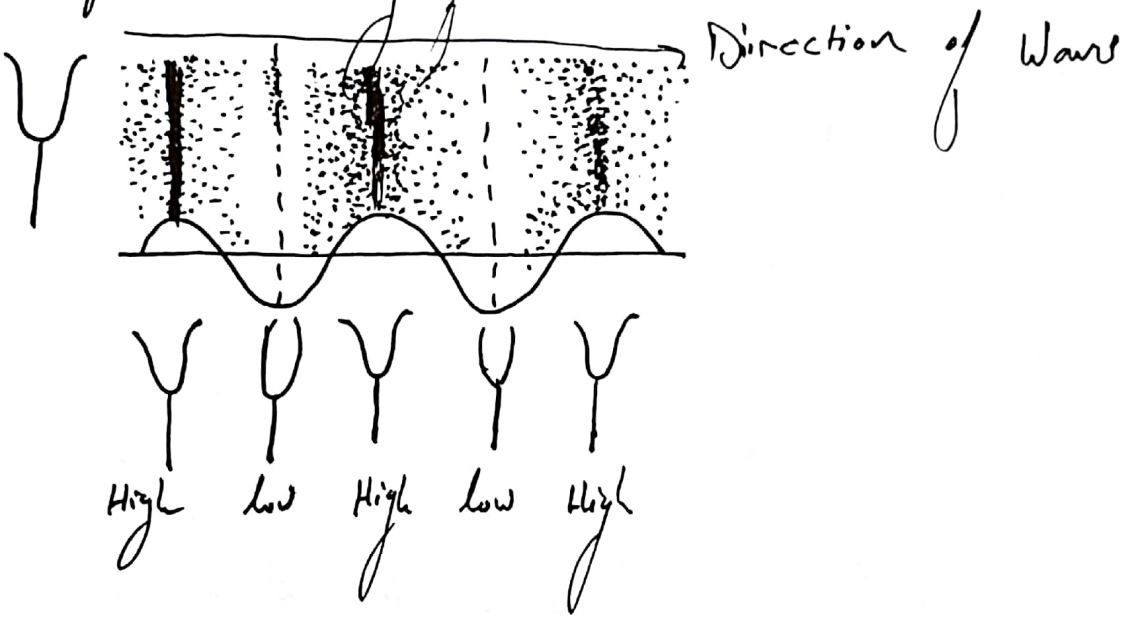


Sound

→ Is a Compression wave

↳ a region of high molecular density and air pressure

Example - A tuning fork



Characteristics of Sound Waves

→ Sound Waves have direction:

→ Motion of the elements of a medium in a longitudinal sound wave is back and forth along the direction in which the wave travels.

→ The net motion of an individual molecule is zero.

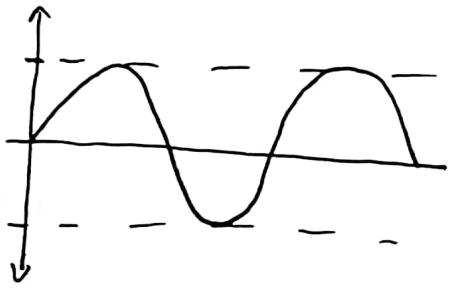
→ the direction of the Compression is the direction of the wave.

→ Audible waves: longitudinal waves between ~20 - 20,000 kHz

audible range

← Back

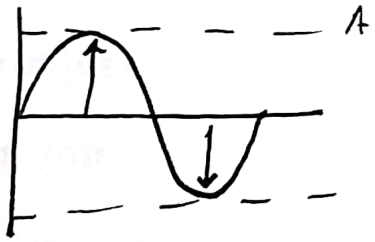
All Waves Can be described as Summation of Sine Waves.



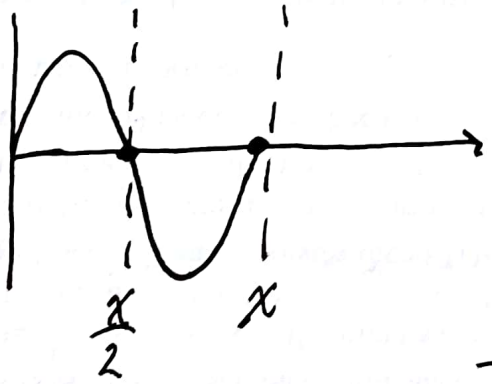
$f(x) = \sin(x)$

→ Mathematical waves have the following properties

↳ Amplitude: distance from y-axis to peak



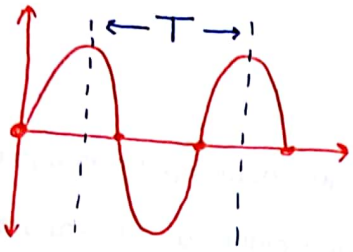
→ Frequency: Number of Cycles per unit "x"



$f = \frac{1\ \text{Cycle}}{x} = \frac{1}{x}$

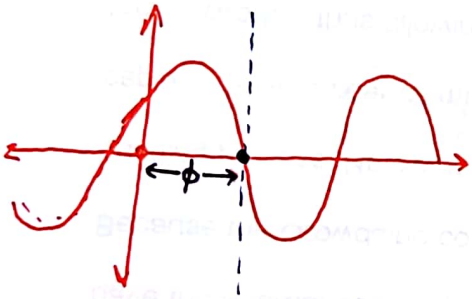
→ if "x" is "time"; $f = \frac{1}{1(s)} = 1\ Hz$

The value at the bottom $\frac{1}{1(s)}$ is the <u>period</u>



$f = \frac{1}{T}$

→ Phase (ϕ): the offset of the f function along the x-axis



→ Putting this together, the general equation for a wave may be written:

~~$f(x) = A\sin(f \cdot \pi \cdot x +$~~

$f(x) = A \cdot \sin(\underbrace{2\pi f}_{\omega} \cdot x + \phi)$

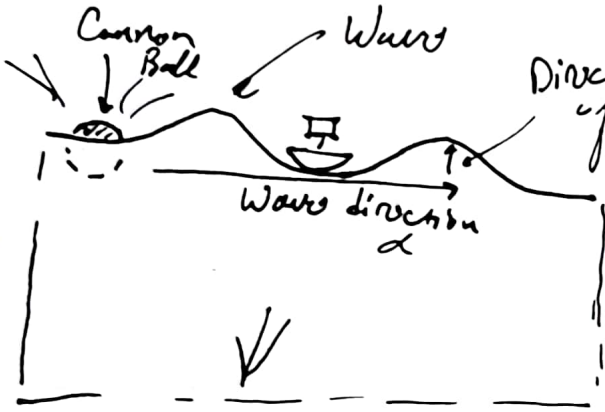
→ $\omega = 2\pi f$; angular frequency (rate of change of function arguments in units of $\frac{rad}{s}$)

→ A sine wave is a mathematical curve that describes a smooth periodic oscillation.

→ It is a perfectly analog concept.

Types of Waves

→ Traveling Waves: a pulse travels through a medium



- Cannon ball splashes into the ocean
→ Waves ripple outward
→ the direction of the wave is perpendicular to the displacement of the water
↳ this is a transverse wave

→ A pressure wave moving through air launches the particles horizontally
↳ If the elements of the medium are displaced in the direction of the wave
→ Longitudinal Wave.

→ Slinky example

Interference of Waves

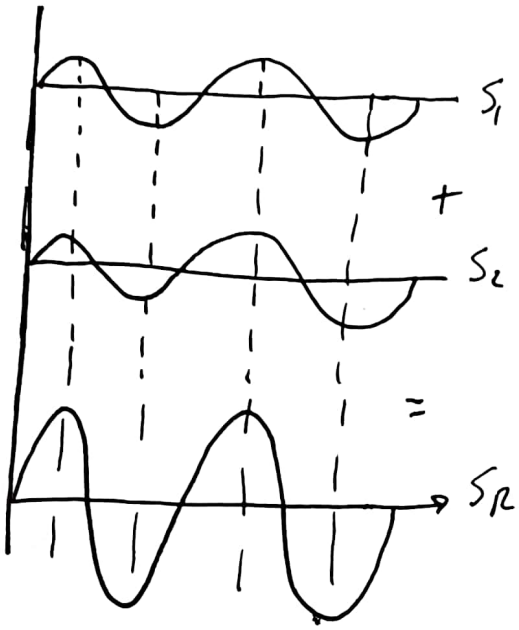
→ Travelling Waves can pass through the same region at the same time.
→ Pebbles in a pond, expanding circular waves pass through each other.
↳ Sound Waves work the same way.

→ Where waves ~~overlap~~ overlap, superposition occurs

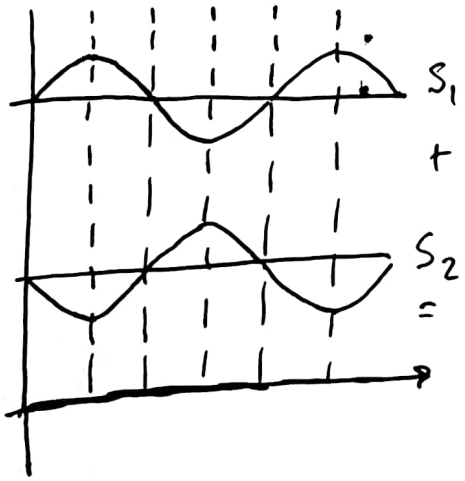
→ Superposition principle
"When two or more travelling waves encounter each other while moving through a medium, the resultant wave is found by adding together the displacements of the individual waves point by point"

Superposition (Interference) in Sine Waves

→ Waves can be added together in a process called "superposition".

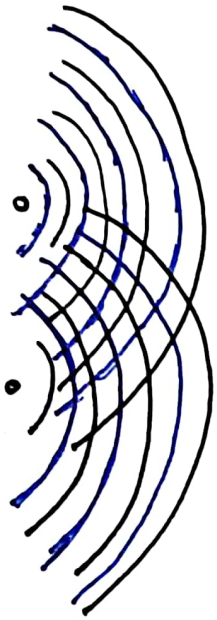


→ Constructive interference
↳ Waves travelling through the same medium add to give a new waveform



→ If the waves are out of "phase", get "destructive interference".

Superposition / Interference in Sound (2D)

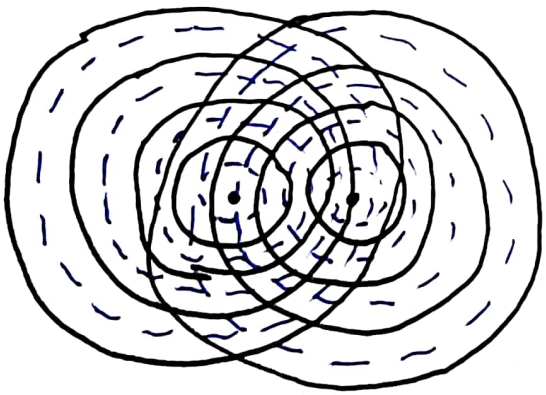


→ 2 point sources of audio spaced appropriately for apart will cause regions of constructive and destructive interference.

→ Where 2-peaks or 2 troughs meet you get a loud spot

→ Where a trough meets a peak you get a quiet spot.

Pebbles in a pond



→ Dropping pebbles in a pond near one another

→ Pebbles displace the surface causing a ripple

→ Ripples carry the energy along the surface

→ Where the peaks meet, they add

→ Where peak meets trough, the cancel out

The Sum of two Waves

→ A sinusoidal wave travelling to the right along the x-axis

$W_1(x,t) = A\cos(kx - \omega t)$ $\quad k = \frac{2\pi}{\lambda}$ $\quad \omega = 2\pi f$

$W_2(x,t) = A\cos(kx - \omega t + \phi)$

→ $W_1 + W_2 = A[\cos(kx - \omega t) + \cos(kx - \omega t + \phi]$

$\cos(a) + \cos(b) = 2\cos\left(\frac{a-b}{2}\right) \cdot \cos\left(\frac{a+b}{2}\right)$

$= A \cdot \left[2\cos\left(\frac{\overset{a}{kx - \omega t} - \overset{b}{kx - \omega t + \phi}}{2}\right)\cos\left(\frac{kx - \omega t + kx - \omega t + \phi}{2}\right)\right]$

$= A \cdot [2\cos(\phi/2) \cdot \cos(kx - \omega t + \frac{\phi}{2})]$

$= 2A\cos(\phi/2) \cdot \cos(kx - \omega t + \frac{\phi}{2})$

→ For even multiples of pi
→ Will not be zero
* yes, it may be negative - but it will not cancel out

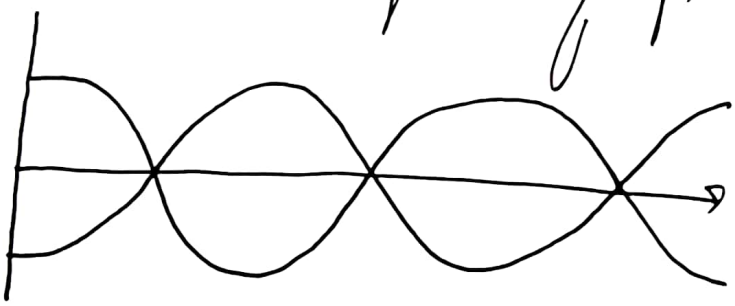
→ $\cos(\frac{2\pi}{2}) = -1$, $\cos(\frac{4\pi}{2}) = 1$; etc

→ $\cos(\frac{\pi}{2}) = \phi$; $\cos(\frac{3\pi}{2}) = \phi$

→ $\cos(\pi/2) = 90°$

→ $\cos(\frac{2\pi}{2}) = 180°$

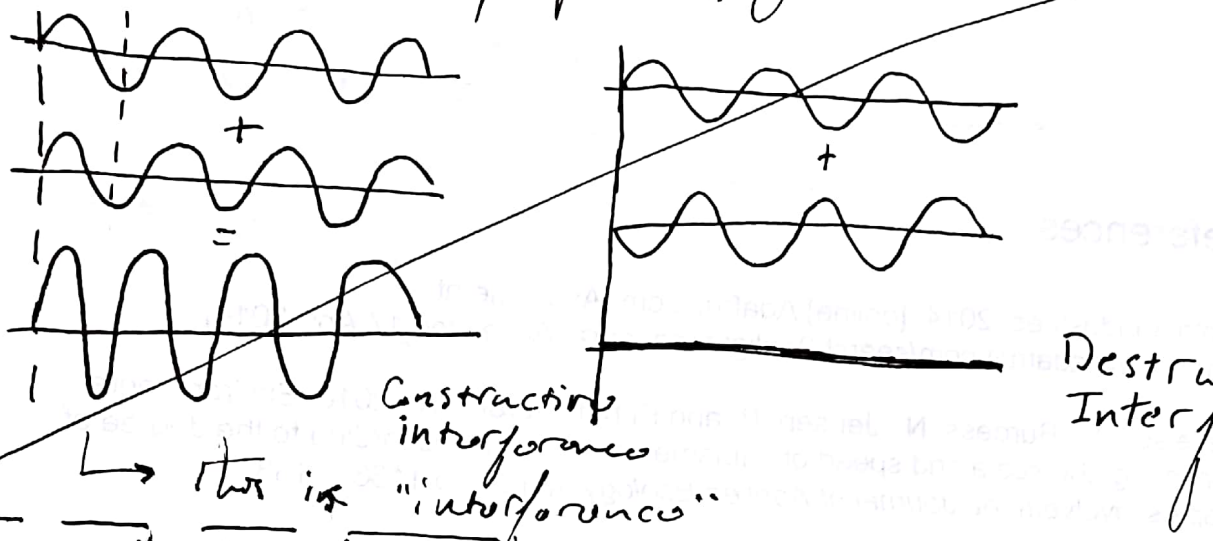
↳ Odd multiples of pi are perpendicular



→ Cancel each other out

→ Even multiples give a non-zero sum and thus interfere Constructively

→ Waves can be added together in a process called "superposition"



Constructive interference

Destructive Interference

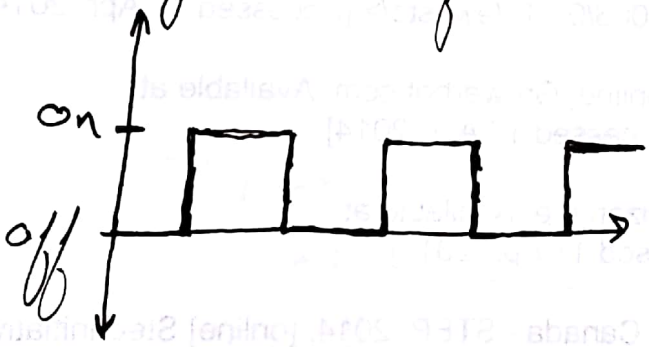
↳ this is "interference"

→ Interference patterns can generate new waveforms

↳ ~~that~~ waveform: shape of a periodic signal (timbre, in musical terms)

→ Square waves, for example

↳ Signals built from binary process



→ Do not exist in "real life" (at least in mechanical waves).

↳ thus, we must ~~can create~~ approximate with sine waves!

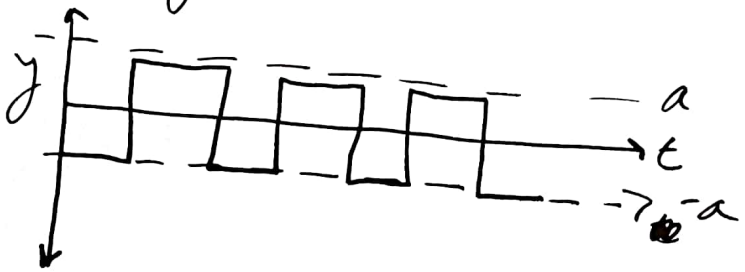
→ $\frac{\sin(1x)}{1} + \frac{\sin(3x)}{3}$ } this is a fourier series

4.5

Fourier Series

→ An expansion of a periodic function as a sum of Sines and Cosines

→ For example
  ↳ Square Wave:



→ We can write as:

$$f(x) = \sum_{n=1,3,5\ldots}^{\infty} \frac{1}{n} \sin(n \cdot x)$$

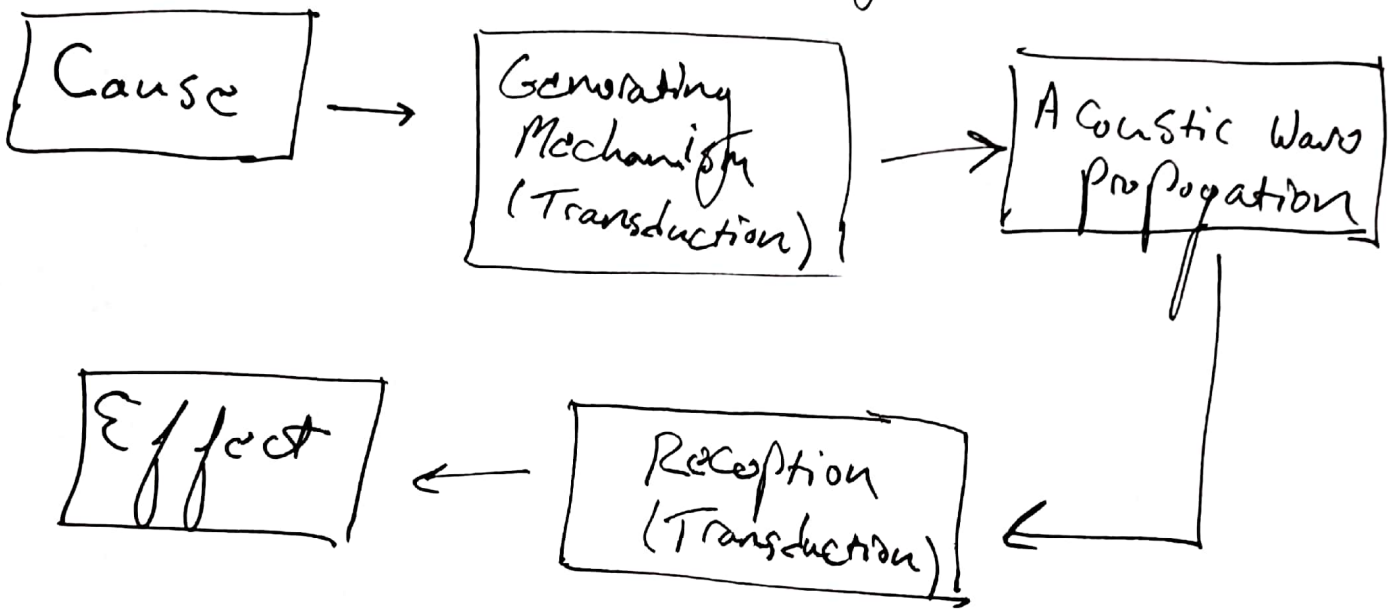
$$= \frac{1}{1}\sin(x) + \frac{1}{3}\sin(3x) + \frac{1}{5}\sin(5x) + \ldots$$

→ Plotting this out ~~we~~ as the number of terms approaches infinity, We find a Square Wave!

→ This may be done for
  ↳ a Saw-tooth Wave
  → triangle Wave etc..

→ this leads us into Signals analysis

The fundamental model of acoustic events:



→ these steps are found in any acoustic event or process

→ Many Causes
- ↳ plucking a string
- → ringing of a bell
- → barking of a dog

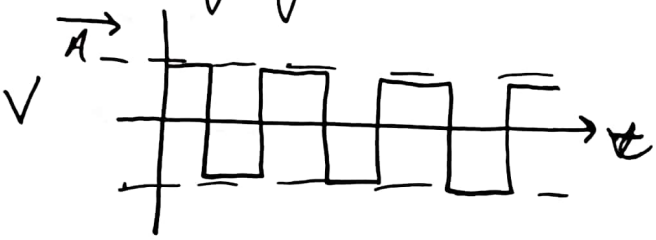
→ these occurances ↳ Something oscillating are transduced to acoustic waves from mechanical motion

↳ you may even go back a step further and consider the reason for the mechanical oscillation
- → thoughts are chemical / electrical

Transduction (A side note)

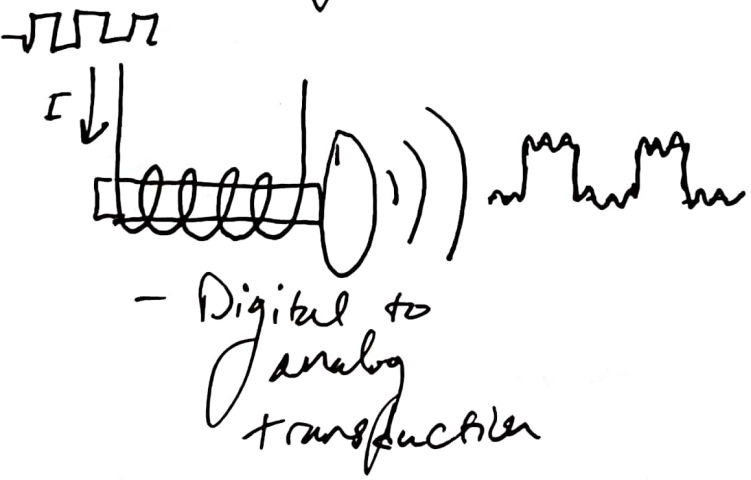
↳ Conversion of one form of energy to another.

→ My definition: Conversion of one information carrying medium to another



→ Consider a binary electronic signal ("on or off")

→ When played through a speaker, the information contained within the square wave ~~theory~~ is converted to a superposition of analog (sound) waves.



- Digital to analog transducer

~~We will return to this in greater detail later...~~

→ Get encoded into language which governs the bio-electrical mechanism of the vocal chords.

→ Energy supplied by the thrust of air from the lungs causes an audible sound interpreted as "Speech" or "song"

→ the resulting mechanical stimulation causes a wave to propagate:

↳ the equation for an acoustic wave is: $\frac{\partial^2 p}{\partial x^2} - \frac{1}{c^2} \cdot \frac{\partial^2 p}{\partial t^2} = 0$

→ Where x is the 1-dimensional propagation direction

→ p is the acoustic pressure

→ c is the speed of sound ($343 \frac{m}{s}$ @ 20°C)

→ If c is constant:

$p = f(ct - x) + g(ct + x)$; f and g are twice differentiable functions

→ the solution is satisfied by allowing one of the functions to be a sinusoid and the other zero - giving us

$$P = P_0 \sin(\omega t \pm kx)$$

→ Plug it in to the original equation... it works!

→ It's a friggin sinusoid!

→ After the propagation of the wave, we have a second transduction - in the case of earthquakes causing a ~~typhoon~~ tidal wave, we wind up with the energy dissipating on the shore.

→ In the instance of me speaking, we wind up with the air molecules bumping into your ear and turning to mechanical vibration of your eardrum and then an electrical signal interpreted by your brain

→ Effect!

→ Acoustics is a rather broad field
↳ We could talk about acoustics of architecture
→ Electro acoustics
→ Bio acoustics
→ Musical acoustics
→ Ultra Sonics
→ Vibration dynamics

→ However, we've established a specific trajectory now - looking at signals and so we will right concentrate on signals analysis

→ Specifically, we will spend quite a bit of time on fourier and spectral analysis.

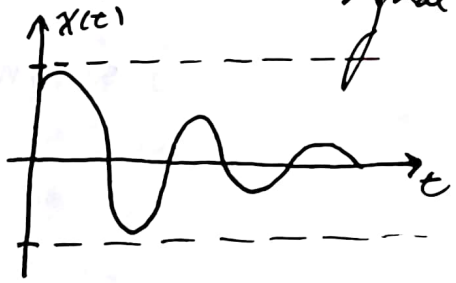
Signal Processing and Analysis

Signal Analysis

→ Signal: "represents a quantity that varies in time"
  ↳ A sound signal is a variation in air pressure over time.
    → Because "sound" is a variation in air pressure,
    → and a "time-signal" varies with time.

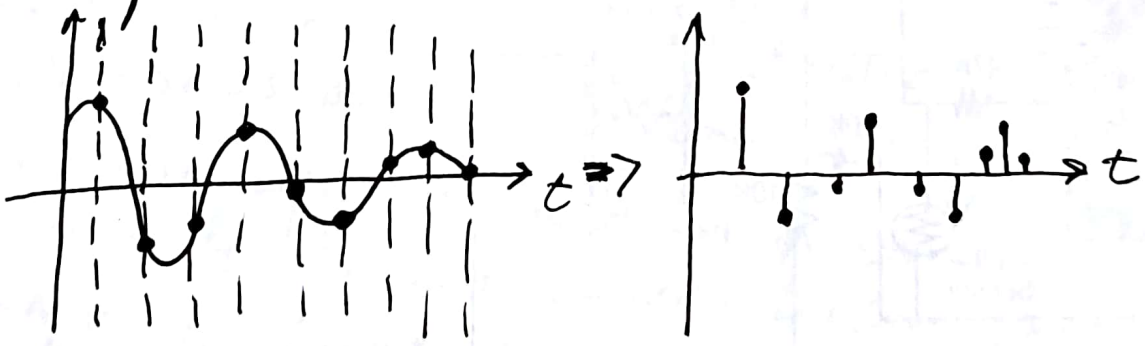
→ Two general types of "time signal":
  ↳ Continuous
  → Discrete

→ Continuous time-signals:
  ↳ Defined along a continuum of time
  → Audible sound that you hear is a continuous signal.



→ Defined for all values in x(t)

→ Discrete time-signals
  ↳ In computer audio (audio inside the computer) time signals are discrete in both time and amplitude.



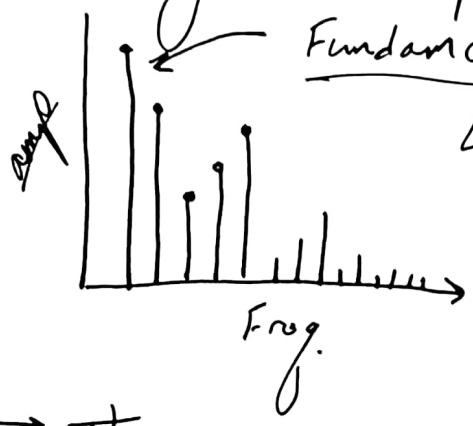
→ Before we get into digital audio, let's define "signal processing"

→ Signal processing:
  → synthesizing, transforming and analyzing signals.
    → Because signals exist independent of medium, these methods may be applied to:
      ↳ Sound signals
      → electronic signals
      → mechanical signals
      → electro-magnetic signals (light)
    → Also apply to signals that vary in space rather than time.
      ↳ topology
    → Multi-dimensional signals
      → Images vary 2-dimensionally across space
      → Movies vary 2-dimensionally across space and 1-dimensionally across time.

→ Returning to the concept of "superposition"
  ↳ the most important concept of signal processing is "spectral decomposition"
  → the reduction of any signal to a sum of sinusoids with different frequencies.

→ the most important algorithm in signal processing is arguably the Fast Fourier Transform (FFT)
  ↳ An efficient way to compute the spectrum of sinusoids

→ Looking at a spectrum plot



Fundamental frequency
↳ lowest frequency component

→ The fundamental frequency also tends to be the dominant frequency
↳ Frequency with largest amplitude.
→ This is not always true

→ The remaining spikes are harmonics of the fundamental frequency
↳ These frequencies are integer multiples of the fundamental

EX → 440 Hz, 880 Hz, 1320 Hz, 1760 Hz, 2200 Hz

→ A fast-fourier transform
↳ produces a plot with frequency along the x-value and amplitude along the y.
→ the lowest frequency is the "fundamental frequency": 440 Hz in "FFT of violin"
→ The largest amplitude frequency is the "dominant frequency."
→ Other spikes are "harmonics" of the fundamental:
↳ In "FFT of violin" 440 Hz is fundamental
→ 880 Hz (A5 - an octave higher than 440 Hz)
→ 1320 Hz (E6; a "perfect fifth" above A5)
→ 1760 Hz (A6; two octaves above fundamental)
→ 2200 Hz (C#7; Major third above A6)
↳ These harmonics make-up A-Major chord
→ Given the frequencies and their amplitudes we may re-construct the signal.

Fourier Transform

→ Is an algorithm that takes a sample of audio and divides into component frequencies.

→ Considered to be one of the most important algorithms ever.

↳ Unfortunately, it's rather opaque.

yikes!
$$\begin{cases} X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i 2\pi k n / N} \\ x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i 2\pi k n / N} \end{cases}$$

→ Deep breath... backing up...

* **What does this do?**
  → Given a signal, find its frequencies

* **How does it do it?**
  → Runs the signal through a series of filters to extract each frequency

* **Why?**
  → It's easier to see what the signal is made of and compare to other signals!

→ To accomplish this, we must set a few ground rules:
  ↳ Filters must only filter out single frequencies
  → Must have a filter for every frequency in the signal or we miss a frequency!
  → the frequencies should be re-combinable
  * smoothing analogy

→ Josep Fourier
  ↳ French guy born in 1768
  → Fourier series
  → Discovered the greenhouse effect
  → Died of heart aneurysms
    ↳ probably exacerbated by falling down a flight of stairs (age 62)

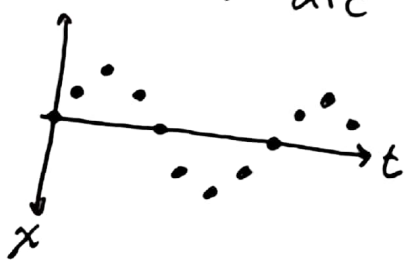
→ Fourier originally dismissed the fundamental idea behind the Fourier transform
  ↳ any signal can be broken down into cycling waves (sines and cosines)

→ Returning to the issue:
  ↳ What does a Fourier Transform do?
    → Takes a signal & finds its "recipe"
    1) Start with a time-based series
    2) Apply filters to find each cyclic function
    3) List out the frequencies
→ Why is this useful?
  ↳ Most relevant:
    → Can represent sound as discrete frequency data
    → Can manipulate and even remove frequencies (Compression → mp3)
→ We can boost frequencies we are interested in.
→ We can apply this to radio signals — if you can separate out frequencies, how can you "tune-in"?
→ Earthquakes are Mechanical Waves
  ↳ We can build buildings that "filter out the dangerous frequencies" of earthquakes.

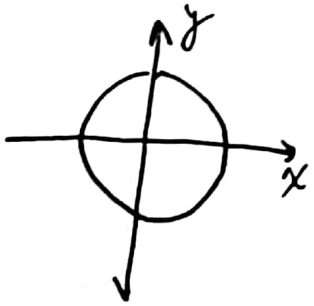
→ To understand the FFT, we can't just think in sinusoids...

→ We must think in terms of "complex sinusoids"
↳ that's a ridiculous way of saying "circles"

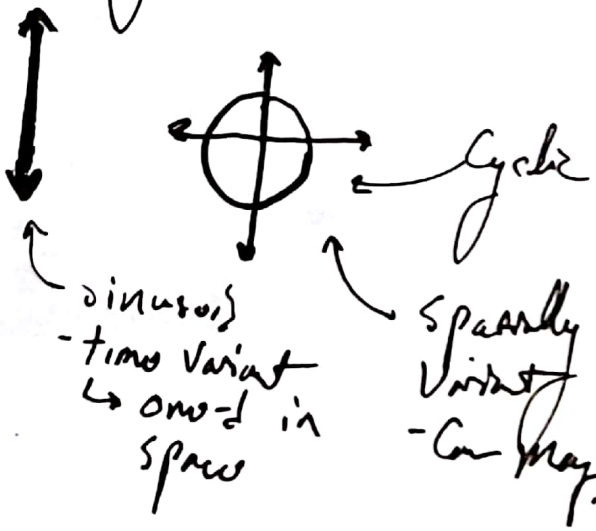
→ Sinusoids are one dimensional in space



→ Circles are two-dimensional in space



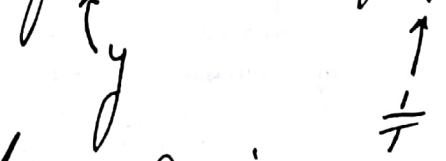
→ you can't describe a cyclic signal in one dimension!
↳ Cyclic signals have:
→ amplitude
→ frequency
→ phase



→ If I want you to draw a circle, I can tell you:
  → How big is the circle (radius, usually)
  → Where you should start (angular displacement)
and - if we are describing drawing the circle in not just two dimensions - but also in TIME - we need speed!

→ What does this give us?
  → Size: amplitude
  → Start position: phase
  → Speed: frequency

Recall: $f(x) = A \cdot \sin(2\pi f \cdot x + \phi)$



→ Holy crap, it's a circle...
  ↳ More concisely, it's a circular path...

→ Each circular path needs:
  → amplitude
  → frequency
  → phase

Musica Universalis

→ Music of the spheres
  ↳ Harmony of the spheres
→ Regards the movement of celestial bodies as a form of Music
  ↳ "Music" there being a harmonic divinity
→ Tolfrush

→ the ancient greeks had an interesting Philosophy long before any of this fourier analysis mumbo jumbo

→ Believed the planets moved in circles around the Earth

↳ A problem with this hypothesis, aside from being extremely incorrect (geocentrism) is that the planets appear to "move backwards" in the sky

→ Ptolomy ~~observ~~ came up with a clever (although also wrong) solution

→ ~~the~~ planets travel in big circles but also travel in little circles at the same time!

→ "theory of epicycles (image)

↳ While this is beautiful and nearly perfectly - to the untrained eye - explains the "Musica Universalis" harmony / music of the spheres

→ We find that by putting circles on top of circles we can generate any orbit

→ Including this once!
↳ homer simpson video

→ In fact, if we can vary the amplitude and frequency of a any collection of orbits, we can generate literally any orbit

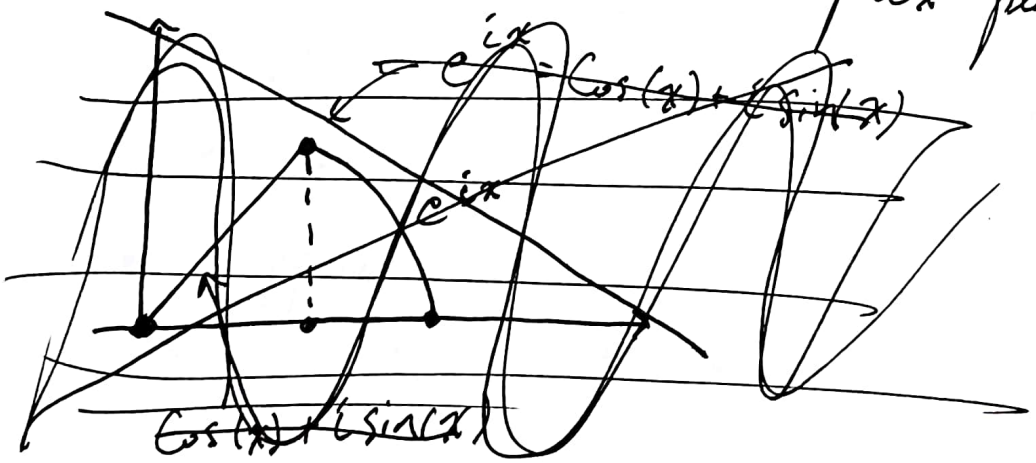
↳ To describe any circular orbit you use the equation:

$$z(t) = Re^{i\omega t}$$

with one variable

→ This is known as "euler's relation"

→ We may use euler's relation to describe a circular path in the complex plane



→ I don't want to get too far into imaginary planes ...

→ all you need to know is that complex numbers are a great way to express 2D concepts with one variable.

→ It is important to note — We are describing moving on a circle — NOT THE GEOMETRY of a circle —
↳ We are describing a path that moves on the circle - if we pick a bunch of values for "t"
→ "cube" example
↳ $10\,e^{(i(\pi/4)t)}$
→ As i progresses we move around a circle!
→ look at circle-two-sine waves
~~↳ Tracking the Mapping the y-component over time gives a moving sin~~
~~→ Because the circle~~
→ mapping the x of the circle-point's position over time to the y-domain gives a cos
→ mapping y position of circle point over time gives sine
→ So, we may write a circle as the sum of these two waves!
→ $\cos(x) + i\sin(x)$

We now have two "Complex" ways to write a circle:

~~Re^{ix}~~ $Re^{i\omega t}$ and $\cos(x) + i\sin(x)$

→ So, $Re^{i\omega x} = \cos(x) + i\sin(x)$

$R = 1$ because

$A =$

let $\omega = \pi$

→ $e^{i\pi} = \cos(\pi) + i\sin(\pi)$

→ since $\cos(\pi) = -1$

and

$\sin(\pi) = 0$

$e^{i\pi} = -1 + 0 \cdot i$

$$\boxed{e^{i\pi} + 1 = 0}$$

Woah!

→ an exponent to an imaginary number and a non-terminating irrational number plus a whole number is... nothing?!

→ this is Euler's identity (and it's awesome!)

→ Back to fourier

$z(t) = Re^{i\omega t}$ describes a point moving along a circle in the complex plane.

→ this produces a simple sinusoidal wave.

$z(t) = R_1 e^{i\omega_1 t} + R_2 e^{i\omega_2 t}$

↳ Describes a point on a circle that orbits the point on the first circle.

→ 2 circles video

→ We can keep doing this indefinitely until...

↳ "Sorcery" video(s)

→ Complex plane video(s)

→ keep adding to infinite circular paths

$$z(t) = \int_{-\infty}^{\infty} R(\omega) e^{i\omega t} \cdot d\omega$$

→ $R(\omega)$ is the fourier transform of $z(t)$

→ If you start by tracing any time-dependent path through 2 dimensions, your path may be perfectly emulated by infinitely many circles of different frequencies:

→ recall, a circular path requires
  ↳ amplitude
  → phase
  → FREQUENCY!

↳ All added up
  → the radii of all these circles is the fourier transform of your path!

~~→ Note: a fourier transform~~

~~→ If we want to write~~

→ Finding the frequencies in an audio signal is ~~basically~~ exactly the same as trying to tease out all the frequencies of our circles —

→ Next we'll delve into how to do this...
  ↳ how to apply filters to tease the circles out of the circles smoothie...

→ Back to our fourier transform equations:

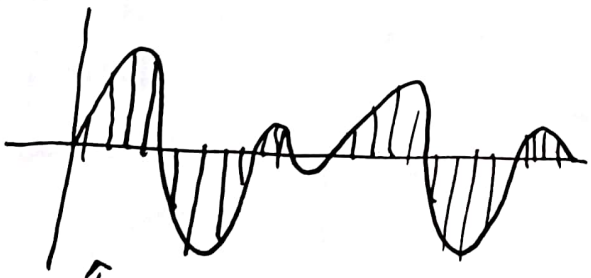
$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$

$$x_n = \frac{1}{N} \sum_{K=0}^{N-1} X_K \cdot e^{i2\pi nk/N}$$

→ These are the discrete and inverse fourier transforms

↳ Discrete because our recording is digital

→ Look closer at a signal



↳ See a bunch of discrete time "spikes"!

→ To figure out what frequencies the signal is composed of, we loop through all of the discrete points and determine the contribution each point gives to each frequency

→ We do this by describing a series of circular paths!

→ Look again at DFT

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}}$$

→ Recall Eulers relation

$z(t) = Re^{i\omega t}$ describes a circular path in the complex plane

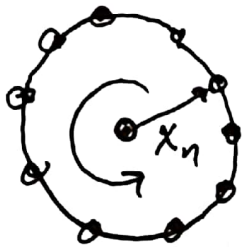
→ the DFT filters through each discrete point in the signal x_n by adding
to describing every possible circular path and adding up the results.

→ Remember, to describe a circular path you need:

↳ amplitude

→ phase

→ frequency



x_n is our radius
↳ points to the values along the circular path
→ K is the frequency
→ 2π is our phase
* Note that the phases is shifted backwards by -2π
~~→ this is so our~~

→ What the analysis is actually doing is:
→ finding a set of cycle speeds, amplitudes and phases that Make a time signal
→ To do this, we try to find a signal that maximally Complements the original signal
↳ Thus we walk backwards to find points to Cancel the original signal!

→ In summation

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}}$$

Means

→ To find the amount of signal that is composed of a specific frequency, spin backwards along a circular path at that frequency and find the average of the points along that path.

→ Moreover, we may return to the original signal:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i 2\pi n k / N}$$

↳ We are literally just doing the same thing, but now we are spinning back through our frequencies to re-produce the points

→ hence "reverse FFT"

→ there's lots of different "Fast Fourier" Algorithms.

↳ Vary in approach to working through the ~~sign~~ circles of signals

→ Differ in approaches with respect to accuracy, speed and application

→ (you will be responsible for researching, explaining and implementing one)!

There's even a Continuous version!

→ That's it! That's the fourier transform!
↳ This insight is Sublimely deep.
→ Is a fundamental concept in pretty much all of music today
↳ Required for digital filters
→ Smoothing / Noise Reduction
→ Bass / Treble boost
→ Effects like "Wah" and Reverb

→ Is also a fundamental concept in pretty much all of our electronics
→ Internet, Phones, Computers, Radios, Satellites, ALL (pretty much) electronic systems require fourier transform algorithms.

→ Solving differential equations

→ Medicine
↳ An MRI literally fourier transforms you by overlaying sinusoidal magnetic waves over you and measuring the resulting electrical interference signatures

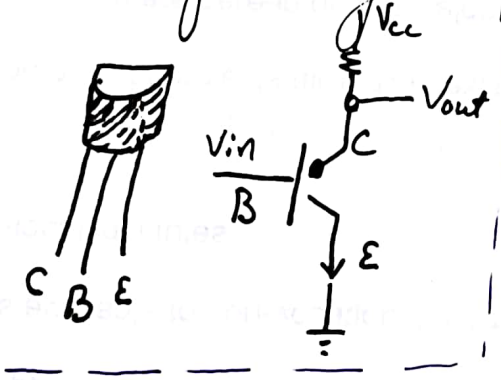
~~Digitization~~

-

Digital Audio

Digital vs Analog

→ The real world is composed of continuous "analog" phenomena.
  ↳ Sound waves are analog.
→ Modern electronic computers are not analog, however!
  ↳ ~~Digital~~ Most electronic computers are DIGITAL!
→ The core mechanism that allows computers to perform logical operations is the transistor.



→ ~~Current~~ ~~voltage~~ flows from V_{cc} to ground if a small amount of current is applied to Vin. (this is an n-p-n type)
→ There are therefore 2 states; "on" and "off."
→ therefore, we have a binary device.
→ Different configurations of transistors give us different logical operations.
→ We can perform these operations as quickly as we can flip an electronic switch (electricity travels at 50%-99% the speed of light)
  ↳ Thus, electronic transistors are a ~~good~~ way fast and reliable way to perform computations

→ Being that computers are built on transistors and transistors are discrete → on or off → computers can only perform discrete computations

→ Computers are digital and sound is analog
↳ To capture, encode, process and decode sound (and information encoded within sound) we need to figure out ways to represent analog functions digitally and produce analog sound from digital information.

→ Information is inherently digital
↳ the basic unit of information is the bit.

→ the process of converting information to a wave and vice versa is modulation / demodulation respectively (mo-dem)

⟹ History of "~~Sound~~ Data over Audio" presentation.

Digital Audio - Sampling

↳ To reduce Continuous-time signals to a discrete-time signal

→ Sample: set of values at a point in time

→ 20-samples.png

↳ Break Continuous signal into regularly spaced discrete values

→ Sampling can be done with any time or space variant function

→ Functions varying in time:

↳ A-to-D Converters turn $x(t)$ to $x[n]$

$x(t) \rightarrow [\text{ADC}] \rightarrow x[n]$

→ $x[n] = x(nT_s)$

↳ Where $x[n]$ is the value of the n^{th} point in the list of n points

↳ T is the "sampling period."

→ Sampling is problematic

↳ We lose information between points!

→ Obviously, the more samples the better...?
  ↳ If human hearing only covers 20 - 20,000 Hz is there a point of sampling higher?

→ Sampling rates:

Use	Rate
Telephone	~8,000 Hz
Voip	16,000 Hz
Human Hearing	20,000 Hz
Wireless Microphones	32,000 Hz
Compressed CDs	37,800 Hz
Standard CD	44,100 Hz
DVD Audio	48,000 Hz
BD Audio	192,000 Hz
DSD	22,579,200 Hz

→ So, if human hearing can only hear frequencies up to about 40,000 Hz why bother with higher rates?

→ Consider two sine waves
  ↳ 2Sines.png
  → as you can see, if sampling is too low, you may do/do more than one sine function that it's.
  → this is called "aliasing"
  → Because you have multiple signals that may describe a sample set, we may mis-identify the signal
    ↳ aliasingSine.png
  → Aliasing is very obvious in some video games
    ↳ aa.png (GTA V)
  → and digital photography
    ↳ AliasingWall.png
→ In audio, vendors sampling causes "lo-fi" sounds.
  ↳ audacity example.

→ For Periodic Sampling, you must sample at twice the frequency of the signal
↳ known as the nyquist rate
→ Periodic sampling is altogether not ideal
↳ you must contend with the aliasing problem
→ the nyquist rate may be hard to find if the signal is composed of many frequencies.
→ Sampling at higher rates takes more space!
→ If not periodic - non-periodic! (non-uniform)
↳ non-uniform.png
↳ Non-uniform sampling avoids this issue - we don't need to worry about harmonic "aliases" sneaking through because the probability of another signal sharing the same point will be low! Sample

→ Of course, non-uniform sampling has its issues:

→ hidden periodicity that may cause aliasing ~~glowy~~ glow

↳ 3.gif

→ Jittering

↳ if we have a pseudo-random sampling period, we may end up with "jitter" when the signal is reconstructed

→ the computer assumes linear jumps rather than smooth curves, (image4.gif)

→ May introduce all sorts of nasty

↳ image8.gif

→ image9.gif

Recall:

↳ Discrete-time signals are represented as a sequence* of numbers

$$x = \{x[n]\}, \quad -\infty < n < \infty$$

where n is an integer

*a sequence is simply a function that has a domain composed of integers.

→ To convert from an analog signal to digital:

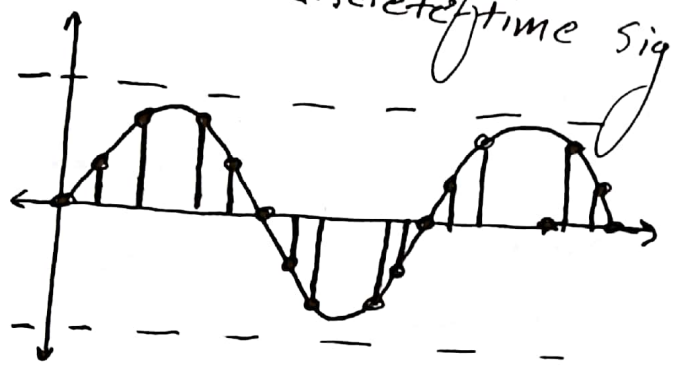
$$x[n] = x_a(n \cdot T), \quad -\infty < n < \infty$$

Where T is the "sampling period"

→ the reciprocal of the sampling period is — of course — the sampling frequency

→ We've discussed some of the issues surrounding periodic sampling — but let's assume that anti-aliasing filters make this work. And large sampling

→ We convert analog to digital because:
a) Computers are digital
b) We can perform interesting operations on discrete-time signal(s)



$x[n] = \{0, 2, 4, 2, 0, -2, -4, -2, \ldots n\}$

→ This is the time signal above written as a sequence

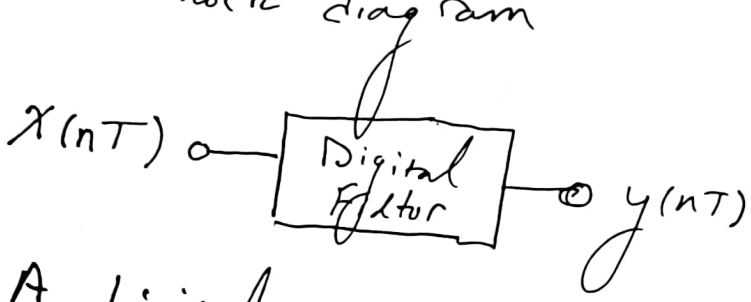
Recall that $x[n] = x_a(n \cdot T)$, $-\infty < n < \infty$

→ With this defined sequences, we may use a digital filter to manipulate the signal

↳ Definition:

"A digital filter is a system which receives a discrete-time signal input and produces a discrete-time signal output."

→ As a block diagram



↳ A digital filter is what engineers call a "transfer function."

~~→ We may look at the~~

→ To better understand this, lets look at some raw discrete audio data.

↳ First, let's discuss a few pieces of software:

→ SoX

↳ Sound eXchange

→ awesome tool for converting audio files between formats

→ Can also do spectrograms

↳ apply filters, etc

→ We will use SoX to convert uncompressed Wav to a human-readable PCM file (.dat)

→ side-note:

PCM stands for "pulse-code modulation"

↳ representation of ~~the~~ analog signal samples as amplitudes of samples at regular intervals

↳ WAV

↳ aiff

↳ PCM

→ Sox signal.wav signal.dat

↳ produces a human-readable .dat file

```
; Sample Rate 44100
; Channels 1
```
} header data

Time Column	Pulses
0	0
1	0.5
⋮	⋮

→ The resulting pulses are our $x[n]$

→ If $x[n]$ is just a list of pulses, we may easily perform a number of operations on it!

Advantages of using digital filters

The following list gives some of the main advantages of digital over analog filters.

1. A digital filter is *programmable*, i.e. its operation is determined by a program stored in the processor's memory. This means the digital filter can easily be changed without affecting the circuitry (hardware). An analog filter can only be changed by redesigning the filter circuit.

2. Digital filters are easily *designed*, *tested* and *implemented* on a general-purpose computer or workstation.

3. The characteristics of analog filter circuits (particularly those containing active components) are subject to drift and are dependent on temperature. Digital filters do not suffer from these problems, and so are extremely *stable* with respect both to time and temperature.

4. Unlike their analog counterparts, digital filters can handle *low frequency* signals accurately. As the speed of DSP technology continues to increase, digital filters are being applied to high frequency signals in the RF (radio frequency) domain, which in the past was the exclusive preserve of analog technology.

5. Digital filters are very much more *versatile* in their ability to process signals in a variety of ways; this includes the ability of some types of digital filter to adapt to changes in the characteristics of the signal.

6. Fast DSP processors can handle complex combinations of filters in parallel or cascade (series), making the hardware requirements relatively *simple* and *compact* in comparison with the equivalent analog circuitry.

Operation of digital filters

In this section, we will develop the basic theory of the operation of digital filters. This is essential to an understanding of how digital filters are designed and used.

Suppose the "raw" signal which is to be digitally filtered is in the form of a voltage waveform described by the function

$$V = x(t)$$

where t is time.

This signal is sampled at time intervals h (the sampling interval). The sampled value at time $t = ih$ is

$$x_i = x(ih)$$

Thus the digital values transferred from the ADC to the processor can be represented by the sequence

$$x_0, x_1, x_2, x_3, \ldots$$

corresponding to the values of the signal waveform at

$$t = 0, h, 2h, 3h, \ldots$$

and $t = 0$ is the instant at which sampling begins.

At time $t = nh$ (where n is some positive integer), the values available to the processor, stored in memory, are

$$x_0, x_1, x_2, x_3, \ldots x_n$$

Note that the sampled values x_{n+1}, x_{n+2} etc. are not available, as they haven't happened yet!

The digital output from the processor to the DAC consists of the sequence of values

$$y_0, y_1, y_2, y_3, \ldots y_n$$

In general, the value of y_n is calculated from the values $x_0, x_1, x_2, x_3, \ldots, x_n$. The way in which the y's are calculated from the x's determines the filtering action of the digital filter.

In the next section, we will look at some examples of simple digital filters.

Examples of simple digital filters

The following examples illustrate the essential features of digital filters.

1. Unity gain filter:

$$y_n = x_n$$

Each output value y_n is exactly the same as the corresponding input value x_n:

$$y_0 = x_0$$
$$y_1 = x_1$$
$$y_2 = x_2$$
$$\ldots etc$$

This is a trivial case in which the filter has no effect on the signal.

2. Simple gain filter:

$$y_n = Kx_n$$

where K = constant.

This simply applies a gain factor K to each input value.

$K > 1$ makes the filter an amplifier, while $0 < K < 1$ makes it an attenuator. $K < 0$ corresponds to an inverting amplifier. Example (1) above is simply the special case where $K = 1$.

3. Pure delay filter:

$$y_n = x_{n-1}$$

The output value at time $t = nh$ is simply the input at time $t = (n-1)h$, i.e. the signal is delayed by time h:

$$y_0 = x_{-1}$$
$$y_1 = x_0$$
$$y_2 = x_1$$
$$y_3 = x_2$$
$$\ldots \text{ etc}$$

Note that as sampling is assumed to commence at $t = 0$, the input value x_{-1} at $t = -h$ is undefined. It is usual to take this (and any other values of x prior to $t = 0$) as zero.

4. Two-term difference filter:

$$y_n = x_n - x_{n-1}$$

The output value at $t = nh$ is equal to the difference between the current input x_n and the previous input x_{n-1}:

$$y_0 = x_0 - x_{-1}$$
$$y_1 = x_1 - x_0$$
$$y_2 = x_2 - x_1$$
$$y_3 = x_3 - x_2$$
$$\ldots \text{ etc}$$

i.e. the output is the *change* in the input over the most recent sampling interval h. The effect of this filter is similar to that of an analog differentiator circuit.

5. Two-term average filter:

$$y_n = \frac{x_n + x_{n-1}}{2}$$

The output is the average (arithmetic mean) of the current and previous input:

$$y_0 = \frac{x_0 + x_{-1}}{2}$$

$$y_1 = \frac{x_1 + x_0}{2}$$

$$y_2 = \frac{x_2 + x_1}{2}$$

$$y_3 = \frac{x_3 + x_2}{2}$$

... etc

This is a simple type of low pass filter as it tends to smooth out high-frequency variations in a signal. (We will look at more effective low pass filter designs later).

6. Three-term average filter:

$$y_n = \frac{x_n + x_{n-1} + x_{n-2}}{3}$$

This is similar to the previous example, with the average being taken of the current and two previous inputs:

$$y_0 = \frac{x_0 + x_{-1} + x_{-2}}{3}$$

$$y_1 = \frac{x_1 + x_0 + x_{-1}}{3}$$

$$y_2 = \frac{x_2 + x_1 + x_0}{3}$$

$$y_3 = \frac{x_3 + x_2 + x_1}{3}$$

As before, x_{-1} and x_{-2} are taken to be zero.

7. Central difference filter:

$$y_n = \frac{x_n - x_{n-2}}{2}$$

This is similar in its effect to example (4). The output is equal to half the change in the input signal over the previous two sampling intervals:

$$y_0 = \frac{x_0 - x_{-2}}{2}$$

$$y_1 = \frac{x_1 - x_{-1}}{2}$$

$$y_2 = \frac{x_2 - x_0}{2}$$

$$y_3 = \frac{x_3 - x_1}{2}$$

... etc

Order of a digital filter

The *order* of a digital filter is the number of *previous* inputs (stored in the processor's memory) used to calculate the current output.

Thus:

1. Examples (1) and (2) above are zero-order filters, as the current output y_n depends only on the current input x_n and not on any previous inputs.
2. Examples (3), (4) and (5) are all of first order, as one previous input (x_{n-1}) is required to calculate y_n. (Note that the filter of example (3) is classed as first-order because it uses one previous input, even though the current input is not used).
3. In examples (6) and (7), two previous inputs (x_{n-1} and x_{n-2}) are needed, so these are second-order filters.

Filters may be of any order from zero upwards.

Digital filter coefficients

All of the digital filter examples given above can be written in the following general forms:

Zero order: $y_n = a_0 x_n$

First order: $y_n = a_0 x_n + a_1 x_{n-1}$

Second order: $y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$

Similar expressions can be developed for filters of any order.

The constants a_0, a_1, a_2, ... appearing in these expressions are called the *filter coefficients*. It is the values of these coefficients that determine the characteristics of a particular filter.

The following table gives the values of the coefficients of each of the filters given as examples above.

Example	Order	a_0	a_1	a_2
1	0	1	-	-
2	0	K	-	-
3	1	0	1	-
4	1	1	-1	-
5	1	$^1/_2$	$^1/_2$	-
6	2	$^1/_3$	$^1/_3$	$^1/_3$
7	2	$^1/_2$	0	$-^1/_2$

SAQ 1 For each of the following filters, state the *order* of the filter and identify the values of its coefficients:

(a) $y_n = 2x_n - x_{n-1}$

(b) $y_n = x_{n-2}$

(c) $y_n = x_n - 2x_{n-1} + 2x_{n-2} + x_{n-3}$

Recursive and non-recursive filters

For all the examples of digital filters discussed so far, the current output (y_n) is calculated solely from the current and previous input values (x_n, x_{n-1}, x_{n-2}, ...). This type of filter is said to be *non-recursive*.

A *recursive* filter is one which in addition to input values also uses previous *output* values. These, like the previous input values, are stored in the processor's memory.

The word *recursive* literally means "running back", and refers to the fact that previously-calculated output values go back into the calculation of the latest output. The expression for a recursive filter therefore contains not only terms involving the input values (x_n, x_{n-1}, x_{n-2}, ...) but also terms in y_{n-1}, y_{n-2}, ...

From this explanation, it might seem as though recursive filters require more calculations to be performed, since there are previous output terms in the filter expression as well as input terms. In fact, the reverse is usually the case: to achieve a given frequency response characteristic using a recursive filter generally requires a much lower order filter (and therefore fewer terms to be evaluated by the processor) than the equivalent non-recursive filter. This will be demonstrated later.

Note

Some people prefer an alternative terminology in which a non-recursive filter is known as an FIR (or Finite Impulse Response) filter, and a recursive filter as an IIR (or Infinite Impulse Response) filter.

These terms refer to the differing "impulse responses" of the two types of filter. The impulse response of a digital filter is the output sequence from the filter when a *unit impulse* is applied at its input. (A unit impulse is a very simple input sequence consisting of a single value of 1 at time $t = 0$, followed by zeros at all subsequent sampling instants).

An FIR filter is one whose impulse response is of finite duration. An IIR filter is one whose impulse response theoretically continues for ever because the recursive (previous output) terms feed back energy into the filter input and keep it going. The term IIR is not very accurate because the actual impulse responses of nearly all IIR filters reduce virtually to zero in a finite time. Nevertheless, these two terms are widely used.

Example of a recursive filter

A simple example of a recursive digital filter is given by

$$y_n = x_n + y_{n-1}$$

In other words, this filter determines the current output (y_n) by adding the current input (x_n) to the previous output (y_{n-1}):

$$y_0 = x_0 + y_{-1}$$
$$y_1 = x_1 + y_0$$
$$y_2 = x_2 + y_1$$
$$y_3 = x_3 + y_2$$
$$\ldots \text{ etc}$$

Note that y_{-1} (like x_{-1}) is undefined, and is usually taken to be zero.

Let us consider the effect of this filter in more detail. If in each of the above expressions we substitute for y_{n-1} the value given by the previous expression, we get the following:

$$y_0 = x_0 + y_{-1} = x_0$$
$$y_1 = x_1 + y_0 = x_1 + x_0$$
$$y_2 = x_2 + y_1 = x_2 + x_1 + x_0$$
$$y_3 = x_3 + y_2 = x_3 + x_2 + x_1 + x_0$$
$$\ldots \text{ etc}$$

Thus we can see that y_n, the output at $t = nh$, is equal to the sum of the current input x_n and all the previous inputs. This filter therefore sums or *integrates* the input values, and so has a similar effect to an analog integrator circuit.

This example demonstrates an important and useful feature of recursive filters: the economy with which the output values are calculated, as compared with the equivalent non-recursive filter. In this example, each output is determined simply by adding two numbers together. For instance, to calculate the output at time $t = 10h$, the recursive filter uses the expression

$$y_{10} = x_{10} + y_9$$

To achieve the same effect with a non-recursive filter (i.e. without using previous output values stored in memory) would entail using the expression

$$y_{10} = x_{10} + x_9 + x_8 + x_7 + x_6 + x_5 + x_4 + x_3 + x_2 + x_1 + x_0$$

This would necessitate many more addition operations as well as the storage of many more values in memory.

Order of a recursive (IIR) digital filter

The *order* of a digital filter was defined earlier as the number of previous inputs which have to be stored in order to generate a given output. This definition is appropriate for non-recursive (FIR) filters, which use only the current and previous inputs to compute the current output. In the case of recursive filters, the definition can be extended as follows:

> **The order of a recursive filter is the largest number of previous input *or* output values required to compute the current output.**

This definition can be regarded as being quite general: it applies both to FIR and IIR filters.

For example, the recursive filter discussed above, given by the expression

$$y_n = x_n + y_{n-1}$$

is classed as being of first order, because it uses one previous output value (y_{n-1}), even though no previous inputs are required.

In practice, recursive filters usually require the *same number of previous inputs and outputs*. Thus, a first-order recursive filter generally requires one previous input (x_{n-1}) and one previous output (y_{n-1}), while a second-order recursive filter makes use of two previous inputs (x_{n-1} and x_{n-2}) and two previous outputs (y_{n-1} and y_{n-2}); and so on, for higher orders.

Note that a recursive (IIR) filter must, by definition, be of at least first order; a zero-order recursive filter is an impossibility. (Why?)

Audio Synthesis

→ Broadly, the creation of sounds by adding together wave-forms (or applying filters).

→ Synthesizer
  ↳ A musical instrument which generates audio signals through electronic means
    → Moog Synthesizer, for example
    → It's difficult to say where and when the earliest synth was created
      ↳ Generally used as a term synonymous with electronic instruments
        → Will talk more about this in the next section

→ Waveform Synthesis
  ↳ What distinguishes one musical instrument or one voice or one sound from another?
    → the Waveform!
    → Broadly speaking, the waveform is the shape of a plot
      ↳ in audio signals it is the shape of an amplitude vs time plot

→ Normally the oscillation of some object generates what you "see" as the waveform given the physical properties of the source and the medium.

→ Synthesizers, on the other hand "build" sounds by adding together waveforms or subtracting away.

→ Additive Synthesis
  ↳ build sounds by adding waveforms
  → Will discuss this at length - not yet

→ Subtractive synthesis
  ↳ Uses filters to mold complex and harmonic-rich sounds to a desired waveform
  → We do this when we speak
    ↳ Vocal folds provide a signal that is relatively complex and harmonic rich
    → Changing the shape of the mouth and throat removes some of the harmonics.
    → "Aah" vs "ooh"

→ "Aaah" is harmonic rich

→ ~~"ohhh"~~ "oooh" is muffles many of the harmonic frequencies
  ↳ Call this an "ooh filter"?

→ Switching back and forth gives a "sweeping filter"
  ↳ "overtone singing"
    → Technically, this adds frequencies!

→ ~~Now~~ When doing "subtractive synthesis" We want to start with a rich signal and cut away.
  ↳ Noise is good for this

→ What is Noise?
  ↳ "an undesirable signal"
  → Crappy definition
  → Sometimes we want noise
    ↳ Can be used to block out background sounds
    → White noise fan for sleep

→ In signal processing
  ↳ White noise is a "random" signal with equal intensity at all given frequencies
  ↳ Has some very desirable applications
    → Random number generation
    → Can help with concentration
    → Therapeutic for tinnitus sufferers
      ↳ Masks the effect of tinnitus
→ Pink noise
  ↳ also called $\frac{1}{f}$ noise
    → frequency spectrum such that power $\propto$ frequency
    → Sound like white noise, but "heavier"
  → Is present in ~~a huge~~ many physical, biological and economic systems
  ↳ Is "ubiquitous"
    → Meteorology fluctuations
    → Electromagnetic radiation from stars!!
    → Almost all electronic devices (flicker)
    → Neural activity
    → DNA stats
    → Heart beat rhythms (important!)

→ You name it - pink noise is there

→ Brown (brownian noise)
  ↳ From brownian motion - "random walk"
  → Stronger at lower frequencies
  → Can be easier to listen to for high-pitch sensitives (like myself)

→ Grey noise
  → Equal frequencies when human hearing is taken into account
    ↳ We hear better at different frequencies.
  → Has the same "heavy" quality as brown and pink noise, but also the higher pitch of white noise.

~~→ Redefining "noise"~~

→ Noise in this sense is a random signal
  ↳ We return to the other definition later

Back to Subtractive Synthesis:

→ Take noise and filter out some of the energy at certain frequencies

~~→ Removing frequencies~~

→ Removing aspects of the noise gives specific sounds

↳ Filters are like a sculptors chisel

→ (of course, as we've discovered, filters can also be applied to non-noise signals to change sound)

Back to filters (at a more general analysis):

→ Filters adjust sound:

↳ in frequency domain

→ Boost and reduce certain frequencies

→ General filter design

→ Filters have the following ~~qualities~~ properties:

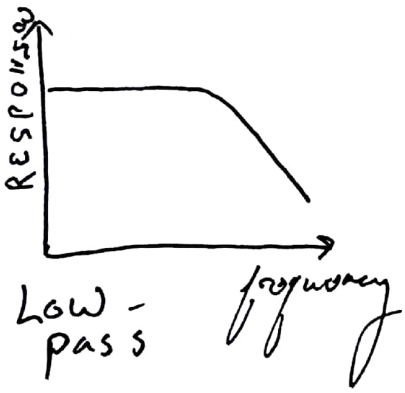
↳ Passband: frequencies to preserve

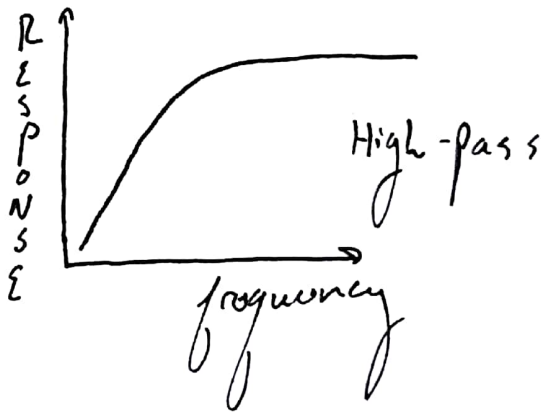
→ Stopband: frequencies to remove

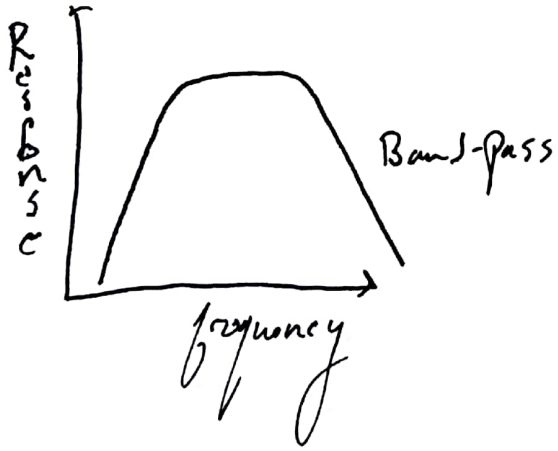
→ Q: (quality) → frequency difference between passband and stopband

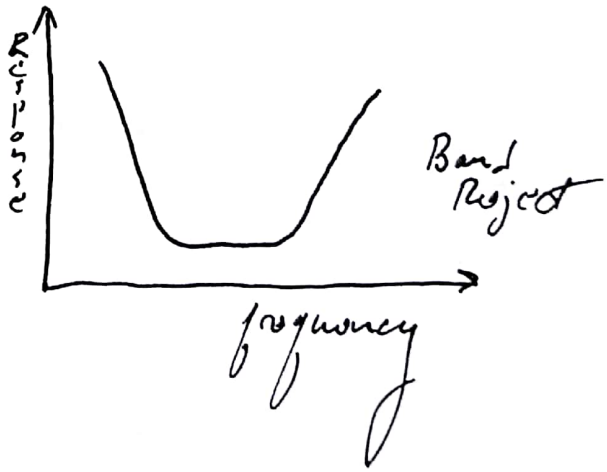
Common Filters:

- High pass:
  - allows only high frequencies
- Low pass:
  - Allows only low frequencies
- Band pass
  - High pass + Low pass
  - → leaves a band









Additive Synthesis

→ Adding sine waves together... sounds familiar?

↳ ... to create timbre

→ The procieved sound quality of a musical note, sound or tone.

→ Distinguishes types of sounds

→ Use fourier analysis to describe the timbre of musical instruments

→ Consider a ~~piano~~ note:

~~→ Loudness, spectral content,~~

→ loudness and spectral content change over time.

↳ and varies from instrument to instrument

→ Considering only the amplitude over time

→ amplitude immediately increases

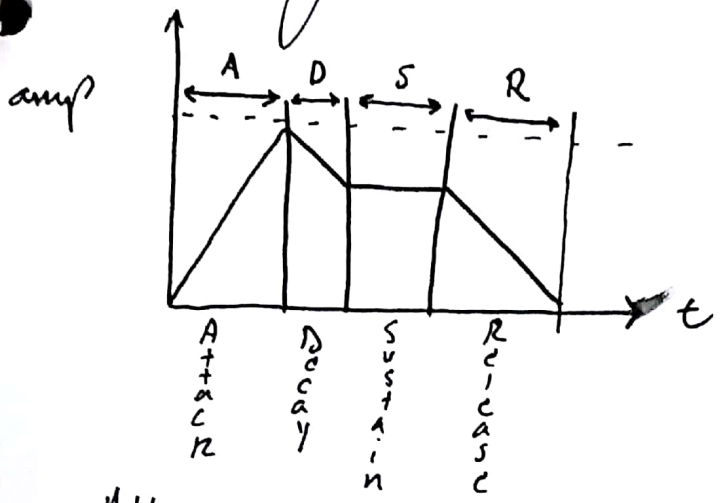
→ decays slightly to designated sustain level

→ Sustains for a time until a key is released

~~→ release time~~

→ time from release to silence.

→ This gives us an "ADSR" envelope



Attack - how long it takes to reach peak amplitude
Delay - time to reach sustained amplitude
Sustain - amplitude at which the note is longest
Release - Decay of amplitude to silence after key is released.

↳ ADSR is applicable to both additive and subtractive approaches

→ We may add up sines until we get the desired envelope.

Audio Data Compression

→ Not to be confused with "dynamic range compression" (dynamically reducing volume range)

→ Data compression
  ↳ Reducing bandwidth and storage requirements for data

→ In digital audio, use various algorithms to achieve this, various "formats" result

↳ Non-Compressed:
  Pulse Code Modulation (PCM)
  ↳ WAV
  → AIFF (Audio interchange file format)
  → AU

→ Lossless data compression
  ↳ Class of algorithms that allow perfect reconstruction of original signal

→ FLAC (free lossless audio Codec)
↳ Algorithm that can compress audio file to 60-70% of the original size.
→ WAV (uncompressed PCM) requires twice as much space!

↳ A FLAC file has four basic components
→ Identification of the stream as flac
↳ literally four bytes "fLaC"
→ Streaminfo block
↳ Info about whole stream
→ sample rate
→ number of channels
→ number of samples
...etc
→ optional metadata not included in Streaminfo
→ Audio frames (samples)

→ The original algorithm was published in 1994 by Tony Robinson

↳ technique called "SHORTEN"

→ Wanted to compress speech databases ~~spoken word~~ to CD ROM

↳ Uses a predicative model of a waveform

→ Linear combination of past samples to predict next sample:

$$\hat{s}(t) = \sum_{i=1}^{p} a_i \, s(t-i)$$

$\hat{s}(t)$ is the predicted sample

$s(t)$ is speech signal

~~such that $e(t) = s(t) - \hat{s}(t)$~~

→ a_i is "coefficient predictor"

3 Main stages of "Shorten" algorithm

→ Blocking

→ Predictative Modelling

→ residual Coding

↳ Blocking:

→ split data into groups and encode each group individually.

↳ If blocks are too small, Compression will be less efficient

→ Too large, will can't generate a good model for prediction.

→ Usually, split up audio into 2,000 - 6,000 samples to define a model.

→ Predictive Modelling

→ Encoder looks for a mathematical function that can represent the block. ~~Only~~

→ Instead of expressing each PCM pair, save the mathematical model for each block

Residual Coding

→ Compare the predicative model to the original signal; take the difference

$e(t) = s(t) - \hat{s}(t)$

error original predicted

↳ the errors themselves are modelled by a probability density function

→ Laplacian or Gaussian

→ This, of course, produces errors

↳ but only ~ 0.004 bits / sample

→ Considered negligable

→ Generally speaking, this is the process for lossless compression in general

→ Generate a model

→ account for discrepancies

Run-length Encoding

- Simple lossless Compression
- Represents Consecutive Runs of data With Value Counts.

Example:

AAAA BBBBB AAA BB AAAA 18 chars

4A 5B 3A 2B 4A 10 chars

→ 44% Compression

- Not useful for highly variant data
  - Many Words end with more data than Start With

Huffman Encoding

1) Take the input and Convert it to an array

Example "TREVOR"

2) Find frequency of each letter and Sort into descending order of freq.

R(2) E(1) O(1) T(1) V(1)

Hoffman Encoding

→ Goto scheme for lots of compression formats

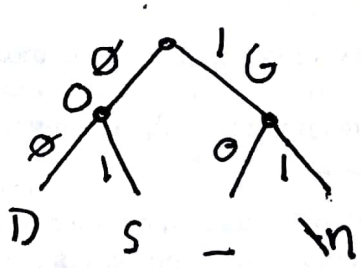
→ Use shorter strings to represent more frequent characters

Example:

good_doggos\n ⇒ 48 bits if each character is represented by an equal-length "word"

→ Instead, we construct a binary tree for frequency of characters:

Char	Freq	Results
G	3	10
O	4	0
D	2	00
S	1	01
_	1	10
\n	1	11



G	3×1	3
O	4×1	4
D	2×2	4
S	1×2	2
_	1×2	2
\n	1×2	2

= 17 bits ~ 65% compression

Note: this is the absolutely most basic approach - there are MANY variations.

→ there are also many other lossless techniques

Lossy Compression

→ Loses data during compression
↳ Cannot reconstruct original signal

→ Lossy compression can create much smaller files than lossless compression

MP3 → Sophisticated compression scheme.

→ 1/10th the space as uncompressed CD audio

→ Reduces quality & removes redundancy

→ Hides reduced quality in parts of the frequency spectrum mastered by louder sounds

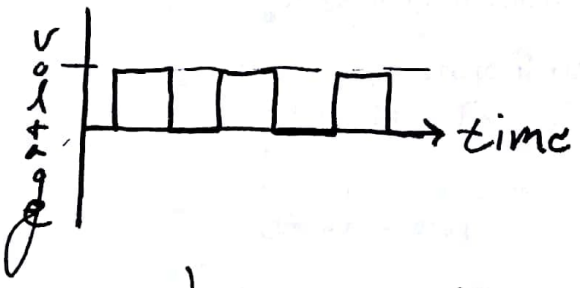
→ Anticipates that sound will be played on lower quality speakers such as headphones.

Computer Audio Hardware

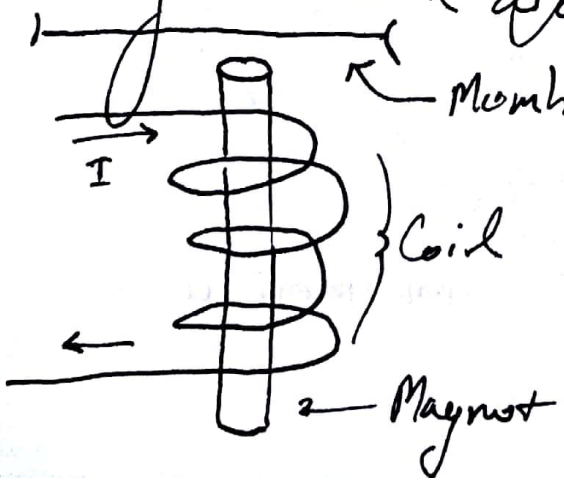
Computer Audio Hardware

→ The first known Computer audio recording:
  ↳ Recorded in 1951 by BBC
  → Generated by Alan Turing;
  ↳ Ferranti Mark 1
  ↳ Did a lot of work on Computer audio in the late 1940's
  → "God Save the Queen"

→ Basic audio is generated by quickly flipping an electronic switch on & off:



→ This oscillating signal is sent through a coil around a magnet



→ Magnet oscillates & bumps a membrane

→ It's almost trivial to make a computer generate a sound.

↳ All you need to do is touch one end of a speaker to a circuit and the other to ground.

→ So long as there is some sort of oscillating signal, you've got sound.

→ An early application of this was, before graphical user interfaces or even CLI's, you could "listen in" to the processor as a program ran

↳ If it buzzes constantly, you knew you were stuck in an infinite loop!

→ An entire sub-culture exists around the ability to create interesting sounds from ~~circuits~~ "short circuits"

→ "Circuit bending"

↳ Read Ghazala video

By definition:

- Circuit bending is:
  - the creative exploration of circuits to create sound
  - Chance-based

→ Prior to "Sound Cards" all computers could do is beep.
  ↳ IBM engineers added a loudspeaker to the 5150
  → This speaker was called the "PC speaker" and was connected directly to the mother-board (invented 1981)
  → Purpose was to give feedback
    ↳ Wave forms generated by a programmable interval timer
      ↳ C8253 (Intel)
        → Used for interrupt requests
        → DRAM refresh
        → PC speaker
→ PC speaker has been (and still is) useful in power-on self-test
  It is always active before the graphics card
  → Provides diagnostic information through "beep codes."
~~→ Back in 1981 IBM stood for "International business machines."~~

→ Some PC speakers were (and are) affixed to the MoBo
  ↳ Many are not
  → simple pin-out
    - SP
    - GND / NC
    - GND
    - +5P
→ The PC speaker is normally only capable of producing a square wave
  ↳ 0 volts low
  → 5 Volts HIGH
→ However
  ↳ Careful timing of pulses and the physics of the speaker allows for a pseudo-actually signal
  → Pulse-Width Modulation
    ↳ Can create the impression of polyphonic music

→ Computers were primarily used for business and only business

↳ But gamers always find a way.

→ In fact, We have video games to thank for most of the advances in computer graphics as well as computer audio.

↳ 1980's were the era of the coin-operated arcade cabinet

↳ and the ~~chip~~ ~~the~~ Computer audio tech that dominated ~~the~~ most of the 80's arcade was the SN76489 digital complex sound generator

→ 3 square wave generators

↳ Wide frequency ranges

~~+ 1 No~~ → 16 volumes

→ 1 Noise generator

↳ White noise and periodic

→ 3 freqs.

→ 16 volume levels

→ Designed for TI-99 Computer
  ↳ Also, BBC Micro
    → Coleco Vision
    → Tandy 1000
→ Games!
  ↳ Mr. Do! Series
  → Road Fighter
  → Time Pilot '84
→ Consoles!
  ↳ Sega System 1 - 2 3 E
  → Neo Geo Pocket / Pocket Color
  → Game Gear
  → Sega Genesis
  → Sega Master System

Much of the rest of the 80's was spent trying to come up with clever ways to make the PC speaker sound more realistic

→ AdLib
↳ 1987
→ Canadian Company founded by Music Professor
→ interfaced YM3812 W/ ISA 8-bit Slot (like a PCI Slot) on a PC and - boom - first "Sound Card" issued.
↳ Unfortunately it had some design issues.
→ Evidence that Creative bribed Yamaha to Sabotage AdLib by sending bad chips
→ Creative Made it to market first
↳ Sound Blaster
→ AdLib Went bankrupt
→ Technically, this was a "Music Card"
↳ Precursor to Modern "Sound Card."
~~→ Music Cards offered basic features:~~
Music Cards were rather basic:
→ Real Sound Could not be played back as the storage requirements
→ Audio Was Synthesized
↳ using the techniques we'll discuss and oscillating Chips

→ they did allow for layering of "instruments" to produce harmonics.
↳ PC speakers couldn't do this - just produce continuing arpeggios.
→ AdLib cards play 11 instruments at a time
→ Music cards were good at tunes - but realistic sounding SFX were basically impossible.

To have realistic computer sounds
→ How to represent analog as 1s and 0s
→ Capture & Conversion ADC
→ How to store
→ How to DAC
→ How to do so cheaply

⟹ 1989, Creative introduced "Sound Blaster"
↳ Copied AdLibs frequency modulation synthesis (literally used the same chips) but added recording and playback
↳ Made it relatively cheap!